DOCUMENT RESUME

ED 057 601 EM 009 445

TITLE	Project Solo; Newsletter Number Six.
INSTITUTION	Pittsburgh Univ., Pa. Dept. of Computer Science.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	2 Dec 70
NOTE	26p.; See also ED 053 566
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	*Computer Assisted Instruction; *Computer Programs; *Programing Languages
IDENTIFIERS	New BASIC (Programming Language); *Project Solo

ABSTRACT

A summary of the current features of the New BASIC System (NBS) as used by Project Solo is presented. A program is given which provides drill-and-practice routine where the random generator output is biased to favor selection of problems on which the student needs most practice. The program shows the use of NBS string functions to find a numerical quantity in a string response. Another program demonstrates the use of multiple statements in NBS. (JY)

PROJECT SOLO
AN EXPERIMENT IN REGIONAL COMPUTING FOR SECONDARY SCHOOL SYSTEMS
NSF/PITT/PPS

University of Pittsburgh • Department of Computer Science • Pittsburgh, Pennsylvania 15213

Newsletter No. 6 December 2, 1970

Teacher Meeting

There will be a meeting at the University of Pittsburgh in 825 Cathedral of Learning on Wednesday, December 16, at 4:00 p.m. The agenda will include (a) A gathering of your comments and suggestions on any changes we should consider in present procedures (except--yes, we know you want more terminals and time--no, we have no extra money!), (b) Some observations from our side of the fence on the usage we have observed, (c) A discussion of what we should propose to the school system and NSF for next year, and the justifications you see for such a request, (d) A showing of a film on H. Dean Brown's work in using computers in learning environments for young children.

Summary of NBS

We are enclosing part of section 9 of the NBS primer. It is a summary of the current features in NBS. As will be apparent from the length of the list, NBS is growing to be an extremely comprehensive language; there will be additional pages for the primer which demonstrate the use of new features coming your way through the newsletter. Later this year you will receive small pocket-size cards summarizing NBS which can be distributed to your students. In the meantime, extra copies of the full page form of the summary are available for insertion in the primer.

Multiple Statements

One of the many unique features of NBS is its ability to handle multiple statements. We are enclosing two sheets numbered 7-2 and 7-3 (also for insertion in the primer) which illustrate the use of such statements. The second example (bottom of page 7-2) is a program that teachers may find useful in posting the results of quizzes, tests, etc. It is on file, and you can call it by typing:

```
-NBS
>RUN 166TD /GRADE/
```

PRINT in FORMAT; New NBS Library Functions

Sheet 8-2 shows two examples using the new "picture" format feature of NBS for precise print control. The full set of format codes is listed on sheet 9-7. Examples of use of library functions recently added to NBS are shown on sheets 6-5, 6-6, and 6-7. Students of probability who wish to use the DICE program on page 6-5 can type:

```
>RUN 104C /DICE/
```

EXAMPLE OF A DRILL AND PRACTICE ROUTINE WHERE THE RANDOM GENERATOR OUTPUT IS BIASED TO FAVOR SELECTION OF PROBLEMS ON WHICH THE STUDENT NEEDS MOST PRACTICE

THIS EXAMPLE ALSO ILLUSTRATES USE OF NBS STRING FUNCTIONS TO FIND THE NUMERICAL QUANTITIES IN A STRING RESPONSE

```
>RUN
DRILL ØN CØMPLEX ARITHMETIC

ALL ANSWERS MUST BE IN THE FØRM 3+5I, THAT IS,
CØNTAIN A REAL PART, A SIGN, AN IMAGINARY
PART, AND THE LETTER I. ALWAYS TYPE 0 FØR ZERØ
EXAMPLES--CØRRECT FØRM: -2+0I    WRØNG FØRM: -2
                         0+3I                3I
                         0+0I                0

WHAT IS THE SUM ØF -3-2I AND -3+8I
?-6+6I
GØØD
WHAT IS THE SUM ØF +4+7I AND -1-5I
?3+2I
RIGHT
WHAT IS THE PRØDUCT ØF +7-4I AND +8-5I
?56+20I
NØ, THE PRØDUCT IS +36-67I
WHAT IS THE PRØDUCT ØF -3+4I AND +5+3I
?-15+7I
NØ, THE PRØDUCT IS -27+11I
WHAT IS THE PRØDUCT ØF +4+7I AND -1+4I
?6-8I
NØ, THE PRØDUCT IS -32+9I
WHAT IS THE SUM ØF -8+8I AND +5+1I
?-3+9I
GØØD
WHAT IS THE PRØDUCT ØF +1+2I AND -6-4I
?7-8I
NØ, THE PRØDUCT IS +2-16I
WHAT IS THE SUM ØF +1-4I AND +6+6I
?7+2I
CØRRECT
WHAT IS THE PRØDUCT ØF +6+0I AND +3-2I
?6-8I
NØ, THE PRØDUCT IS +18-12I
WHAT IS THE PRØDUCT ØF +3-2I AND +2+8I
?6-8I
NØ, THE PRØDUCT IS +22+20I
YØU HAD  4 ADDITIØN PRØBLEMS CØRRECT,
YØU HAD  0 MULTIPLICATIØN PRØBLEMS RIGHT.

YØUR GRADE IN ADDITIØN IS  100%
YØUR GRADE IN MULTIPLICATIØN IS  0%

ØVERALL GRADE IS  40.0%

SØ LØNG
```

Note that the program starts to favor multiplication after it is evident that the student can handle addition.

The grades at the end effectively prescribe special tutoring for him in multiplication.

A more advanced version of this program automatically branches to such tutoring on the basis of this history.

```
>LISTNH

5 VAR=ZERØ
10 A$="'WHAT IS THE SUM ØF ' SD SD'I AND ' SD SD'1'/"
20 B$="'NØ, THE SUM IS ' SØD SØD'1'/"
30 C$="'WHAT IS THE PRØDUCT ØF ' SD SD'I AND ' SD SD'I'/"
40 D$="'NØ, THE PRØDUCT IS ' SØØD SØØD'I'/"
110 PR. "DRILL ØN CØMPLEX ARITHMETIC"
112 PR. PR."ALL ANSWERS MUST BE IN THE FØRM 3+5I, THAT IS,"
114 PR."CØNTAIN A REAL PART, A SIGN, AN IMAGINARY"
115 PR. "PART, AND THE LETTER I. ALWAYS TYPE 0 FØR ZERØ"
117 PR."EXAMPLES--CØRRECT FØRM: -2+0I    WRØNG FØRM:  -2"
118 PR."                        0+3I                  3I"
119 PR."                        0+0I                  0" PR.
120 LET N=N+1
125 IF N>10 LET N=N-1 GØTØ 500
130 R=NUM(10) LET A=NUM(19)-10 LET B=NUM(19)-10
LET C=NUM(19)-10 LET D=NUM(19)-10
140 IF R<(7+M-S) GØTØ 300 ELSE GØTØ 400
300 PRINT IN FØRM A$:A,B,C,D
310 LET E=A+C LET F=B+D
320 GØSUB 600
370 IF ABS(E-R)<.001 AND ABS(F-I)<.001 CALL REIN,
LET S=S+1 GØTØ 120
380 PRINT IN FØRM B$:E,F LET W=W+1 GØTØ 120
400 PRINT IN FØRM C$:A:B:C:D
410 LET E=A*C-B*D LET F=A*D+B*C
420 GØSUB 600
470 IF ABS(E-R)<.001 AND ABS(F-I)<.001 CALL REIN,
LET M=M+1 GØTØ 120
480 PRINT IN FØRM D$:E,F GØTØ 120
500 PR. "YØU HAD ":S:" ADDITIØN PRØBLEMS CØRRECT,"
510 PR. "YØU HAD ":M:" MULTIPLICATIØN PRØBLEMS RIGHT."
520 PR. PR. "YØUR GRADE IN ADDITIØN IS ":(S*100.)/(S+W):"%"
530 PR. "YØUR GRADE IN MULTIPLICATIØN IS ":(M*100.)/(N-S-W):"%"
540 PR. PR. "ØVERALL GRADE IS ":((S+M)*100.)/N:"%"
550 PR. PR. "SØ LØNG"
560 END
600 INPUT R$
610 LET R=VAL(R$)
615 LET R$=RIGHT(R$,LENGTH(R$)-1)
620 LET X1=INDEX(R$,"+") LET X2=INDEX(R$,"-")
630 LET X=MAX(X1,X2)
640 LET I=VAL(RIGHT(R$,LENGTH(R$)-X+1))
650 RETURN
```

L 140 REMARKS: The formula 7+M-S initially biases the random selection of addition problems; as a favorable response to this kind of problem builds up (S increasing), the bias shifts to multiplication. M provides the converse effect for multiplication.

SUBROUTINE 600 extracts the numerical value of the real and imaginary parts of the response R$ which is initially accepted as a string.

```
LØGØUT
```

USING THE RANDOM GENERATOR "NUM" FOR INDEX CONTROL

The following example shows how NUM can be used to set indices (X and Y in this case),so that information stored in arrays is randomly selected. As the example shows, this information can be alphanumeric, since NBS allows string arrays

```
2 A$(1)="#       #" LET C$(1)=A$(1) LET B$(2)=A$(1) LET B$(4)=A$(1)
4 A$(2)="#.      #" LET A$(3)=A$(2)
6 B$(1)="#   .   #" LET B$(3)=B$(1) LET B$(5)=B$(1)
8 C$(3)="#      .#" LET C$(2)=C$(3)
10 A$(4)="#.     .#" LET C$(4)=A$(4) LET A$(5)=C$(4)
12 C$(5)=A$(4) LET A$(6)=A$(4) LET B$(6)=A$(4) LET C$(6)=A$(4)
40 PR."DICE TØSS--HØW MANY TØSSES DØ YØU WISH TØ SEE":
41 INPUT N
42 FØR I=1 TØ N
43 LET X=NUM(6)
44 LET Y=NUM(6)
45 PR. PR.
50 PR."########","########"
55 PR. A$(X),A$(Y)
60 PR. B$(X),B$(Y)
65 PR. C$(X),C$(Y)
70 PR. "########",PR."########",PR.X+Y
75 NEXT I
100 END
```

```
>RUN

DICE TØSS--HØW MANY TØSSES DØ YØU WISH TØ SEE: 4

#######         #######
#     #         #.   .#
#  .  #         #     #
#     #         #.   .#
#######         #######         5


#######         #######
#.   .#         #.    #
#     #         #     #
#.   .#         #    .#
#######         #######         6


#######         #######
#     #         #.   .#
#  .  #         #.   .#
#     #         #.   .#
#######         #######         7


#######         #######
#.   .#         #     #
#  .  #         #  .  #
#.   .#         #     #
#######         #######         6
```

USE OF MOD(X,Y) [= the remainder from the division X/Y]

Example 1:

```
10 REM 'MØD' USED TØ 'PULL ØFF' LAST DIGITS ØF A NUMBER
20 X=NUM(999) PR.X;MØD(X,100);MØD(X,10)  FØR I=1 TØ 10
30 END


>RUN
 414    14     4
 299    99     9
 150    50     0
 63     63     3
 292    92     2
 360    60     0
 680    80     0
 809    9      9
 936    36     6
 612    12     2
```

Example 2:

```
10 REM MØD FCN USED FØR CYCLIC CØNTRØL
20 PR. "TYPE AN INTEGER":
30 INPUT N
40 LET N=INT(ABS(MØD(N,15)))
50 REM NØ MATTER WHAT THE USER TYPES,IT IS CØNVERTED
51 REM TØ A PØSITIVE INTEGER N, WHERE 0<=N<=14
55 PR."------------------------"
56 PR." K";" X";" Y";" A$";"(N=":N:")"
57 PR."------------------------"
60 FØR K=1 TØ N
70 LET X=MØD(K,3)  LET Y=MØD(K-1,3)+1
75 REM 'Y' WILL TAKE ØN THE VALUES 1,2,3 CYCLICALLY AS K=1,2,...N
80 ØN Y GØSUB 201,202,203
90 PR. K;X;Y;A$
100 NEXT K
110 END
201 LET A$="ØH" RETURN
202 LET A$="YØU" RETURN
203 LET A$="KID" RETURN


>RUN
TYPE AN INTEGER?-56.3
------------------------
 K      X      Y      A$    (N= 11)
------------------------
 1      1      1     ØH
 2      2      2     YØU
 3      0      3     KID
 4      1      1     ØH
 5      2      2     YØU
 6      0      3     KID
 7      1      1     ØH
 8      2      2     YØU
 9      0      3     KID
 10     1      1     ØH
 11     2      2     YØU
```

EXAMPLES USING THE STRING FUNCTIONS
VAL, INDEX, LEFT, RIGHT, LENGTH, AND SUBSTR

Example 1

```
10 PRINT "START"                    !REMARKS MADE AFTER
20                                  !A '!' SYMBØL
30                                  !WILL LIST ØNLY
40 C$="67.5 GRAMS"
50 C=VAL(C$)                        !VAL(C$) RETURNS THE NUMERIC
60 IF C>50 LET M=C-50               !VALUE ØF NUMERIC SYMBØLS IN C$
70 PR. C;M;C$                       !ØCCURING BEFØRE THE 'G' IN GRAMS
80 END                              !ALLØWING CALCULATIØN WITH C


>RUN
START
 67.5        17.5        67.5 GRAMS
```

Example 2

```
>10 INPUT A$
>20 X = INDEX(A$," ")
>30 PRINT X
>40 B$ = LEFT(A$,X)
>50 PRINT B$
>60 END
>RUN

?WHØ AM I
 4
WHØ␣

>35 X = X-1
>RUN
?WHAT ARE YØU
 5
WHAT
```

INDEX(A$," ") returns the position of the character between quotes (in this case a space) in the string A$

In the first input example, the INDEX is 4, since the first space in WHO AM I is in the 4th position.

The function LEFT(A$,X) returns the first X characters of A$. RIGHT(A$,X) would return the last X characters.

Example 3

```
>10 PR."WHAT IS YØUR NAME": INPUT N$
>20 LET N=LENGTH(N$)
>30 LET M$=SUBSTR(N$,INT(N/2),3)
>40 PR."GØØD GRIEF! THE 3 MIDDLE LETTERS "
>50 PR."ØF YØUR NAME SPELL ";M$
>RUN

WHAT IS YØUR NAME?HØRATIØ
GØØD GRIEF! THE 3 MIDDLE LETTERS
ØF YØUR NAME SPELL RAT
```

N=7 for HORATIO. M$ contains 3 characters from N$ starting at position INT(N/2).

MULTIPLE STATEMENTS IN NBS

NBS allows several statements to be placed on a line. Use of this feature can make programs more compact, more readable, and more efficient. Square brackets are used in multiple statements to limit the scope of iterations controlled by suffixes.

EXAMPLE 1: Here is a one-line program to print all possible products of integral powers of 2 up to 8*8.

```
-NBS
>10 I=2 [J=2 [PR.I,J,I*J LET J=J*2 WHILE J<9] LET I=I*2 WHILE I<9]

>RUN
2             2             4
2             4             8
2             8             16
4             2             8
4             4             16
4             8             32
8             2             16
8             4             32
8             8             64
```

```
10 REM SAME PROGRAM IN OLD BASIC
20 LET I=2
40 LET J=2
50 PRINT I,J,I*J
60 LET J=2*J
70 IF J<9 GOTO 50
80 LET I=2*I
90 IF I<9 GOTO 40
100 END
```

EXAMPLE 2: The following program plots a bar graph showing the distribution of test grades by percentiles. It also gives the raw mean, and the mean based on 100%. Line 170 illustrates the ability of multiple statements to alter the return of GOSUB (to 230 instead of to 180 in this case).

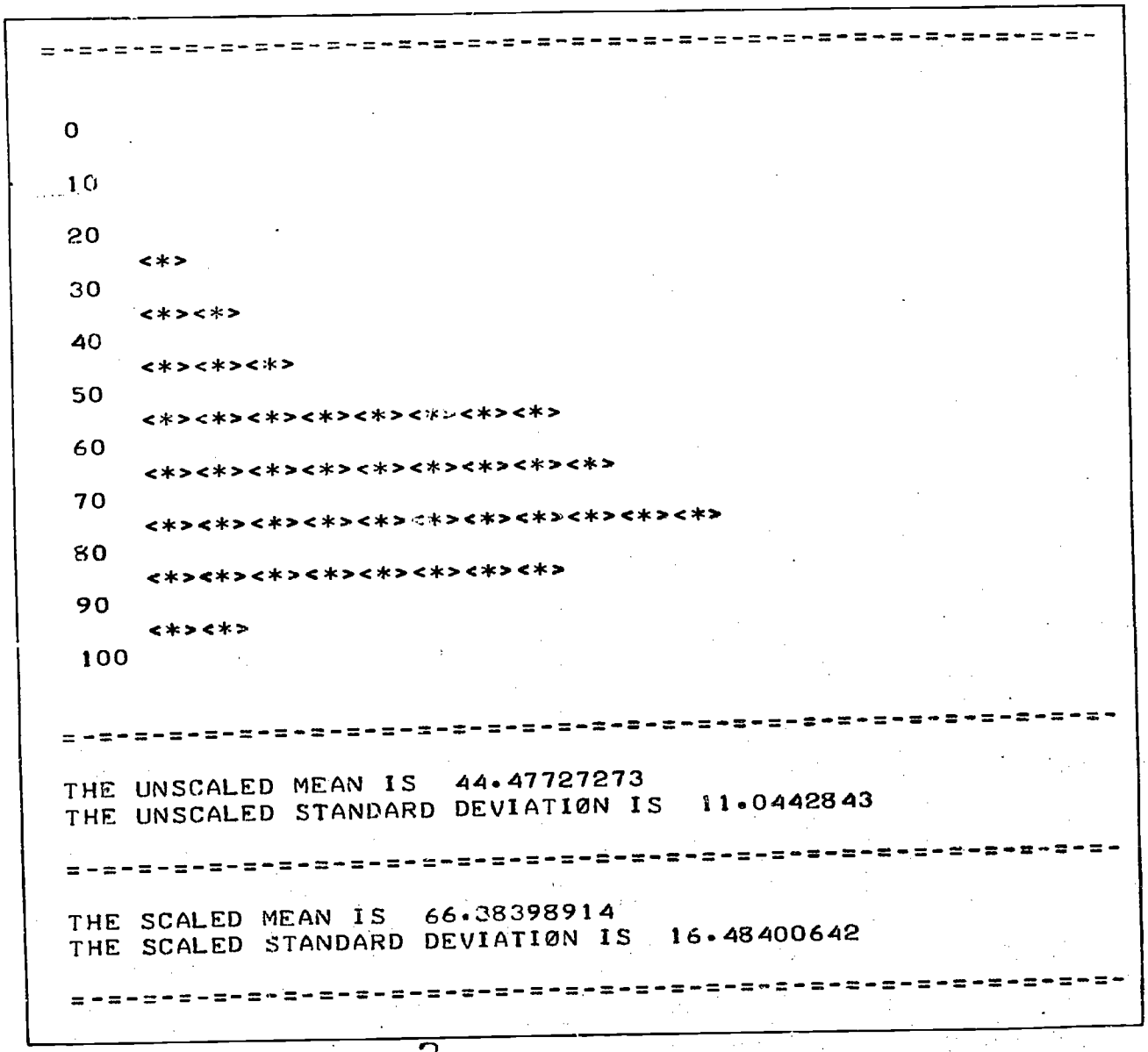
```
100 VAR=ZERØ DIM P(10)
110 PR."WHAT IS MAXIMUM GRADE PØSSIBLE":
120 INPUT M
130 PR. "ENTER A GRADE AFTER EACH '?'."
140 PR. "ENTER 9999 WHEN FINISHED."
150 PR.
160 INPUT G
170 IF G>9000 GØSUB 410 GØTØ 230
180 IF G>M PRINT "GRADE EXCEEDS MAX." GØTØ 160
190 LET R1=R1+G LET R2=R2+G*G
200 LET G=G*100/M LET S1=S1+G LET S2=S2+G*G
210 IF G>99.9 LET G=G-1
220 LET G=G/10+1 LET P(G)=P(G)+1 LET N=N+1 GØTØ 160
230 PR. PR.I
240 FØR I=1 TØ 10
250 PR. TAB(5): [PR."<*>":FØR J=1 TØ P(I)] PR.
270 PR.    I*10
290 NEXT I
300 PR. GØSUB 410
320 PR. "THE UNSCALED MEAN IS ":R1/N
330 D=SQRT((R2-(R1*R1/N))/(N-1))
340 PR. "THE UNSCALED STANDARD DEVIATIØN IS ":D
350 GØSUB 410
370 PR. "THE SCALED MEAN IS ":S1/N
380 D=SQRT((S2-(S1*S1/N))/(N-1))
390 PR. "THE SCALED STANDARD DEVIATIØN IS ":D
395 GØSUB 410
400 END
410 PR. [PR."=-":FØR Q=1 TØ 30] PR. PR. RETURN
```

```
!NØTE: ØNE EXTRA
!'PR.' IS NEEDED
!TØ CANCEL THE
!EFFECT ØF THE
!LAST ':' GENER-
!ATED IN 'FØR'
!LØØPS ØF THIS
!TYPE.
```

Sample Run:

```
WHAT IS MAXIMUM GRADE PØSSIBLE?67
ENTER A GRADE AFTER EACH '?'.
ENTER 9999 WHEN FINISHED.

? 23
? 34
? 36
? 41
? 46
? 78
GRADE EXCEEDS MAX.
? 50
? 28
? 58
? 18
? 24
? 28
? 30
? 42
? 39
? 45
? 48
? 49
? 40
? 50
? 56
? 58
? 41
? 42
? 57
? 58
? 59
? 53
? 61
? 62
? 47
? 34
? 38
? 41
? 42
? 47
? 51
? 53
? 59
? 57
? 34
? 39
? 42
? 48
? 49
? 9999
```



EXAMPLES USING "PRINT IN FORM"

Example 1:

```
10 LET X=6.08324956
20 A$="SD.DDDDDBBBB -D.DDBBBBBBB $***.**BBBBB DDD,DDD,DDD/"
30 PRINT IN FØRM A$:X,X,X,X*1E8
40 REM NØTE VARIABLES IN PRINT STATEMENT MATCH FIELDS IN A$
50 REM SPACES SEPARATE FIELDS IN A$. SLASH / AT END CAUSES LINE FEED

 RUN
+6.08325      6.08          $**6.08       608,324,956
```

Example 2: Note that 2(SD.DDB) is equivalent to SD.DDB SD.DDB

```
100 A$="2(SD.DDB) 2(SD.DDDB) 3(SDD.DDDB) SDD.DDDB/"
110 B$="2(SD.DDB) 2(SD.DDDB) ' UNDEF   ' SDD.DDDB ' UNDEF   ' SDD.DDDB/"

120 C$="2(SD.DDB) 2(SD.DDDB) SDD.DDDB ' UNDEF   ' SDD.DDDB' UNDEF   '/"
130 R=1
140 PRINT " X      Y      SIN     CØS     TAN      CØT      SEC      CSC"
150 FØR X=1 BY -.1 WHILE X>-1.1
160 IF ABS(X)>=1 GØTØ 280
170 Y=ABS(SQRT(R*R-X*X))
180 IF ABS(X)<.005 GØTØ 250
190 IF ABS(Y)<.005 GØTØ 280
200 PRINT IN FØRM A$:X:Y:Y/R:X/R:Y/X:X/Y:R/X:R/Y
210 NEXT X
230 END
240 GØTØ 140
250 X=0
260 PRINT IN FØRM B$:X:Y:Y/R:X/R:X/Y:R/Y
270 GØTØ 210
280 Y=0
290 PRINT IN FØRM C$:X:Y:Y/R:X/R:Y/X:R/X
300 GØTØ 210
```

```
>RUN

 X      Y      SIN     CØS     TAN      CØT      SEC      CSC
+1.00  +0.00  +0.000  +1.000  +00.000   UNDEF   +01.000   UNDEF
+0.90  +0.44  +0.436  +0.900  +00.484  +02.065  +01.111  +02.294
+0.80  +0.60  +0.600  +0.800  +00.750  +01.333  +01.250  +01.667
+0.70  +0.71  +0.714  +0.700  +01.020  +00.980  +01.429  +01.400
+0.60  +0.80  +0.800  +0.600  +01.333  +00.750  +01.667  +01.250
+0.50  +0.87  +0.866  +0.500  +01.732  +00.577  +02.000  +01.155
+0.40  +0.92  +0.917  +0.400  +02.291  +00.436  +02.500  +01.091
+0.30  +0.95  +0.954  +0.300  +03.160  +00.314  +03.333  +01.048
+0.20  +0.98  +0.980  +0.200  +04.899  +00.204  +05.000  +01.021
+0.10  +0.99  +0.995  +0.100  +09.950  +00.101  +10.000  +01.005
+0.00  +1.00  +1.000  +0.000   UNDEF   +00.000   UNDEF   +01.000
-0.10  +0.99  +0.995  -0.100  -09.950  -00.101  -10.000  +01.005
-0.20  +0.98  +0.980  -0.200  -04.899  -00.204  -05.000  +01.021
-0.30  +0.95  +0.954  -0.300  -03.180  -00.314  -03.333  +01.048
-0.40  +0.92  +0.917  -0.400  -02.291  -00.436  -02.500  +01.091
-0.50  +0.87  +0.866  -0.500  -01.732  -00.577  -02.000  +01.155
-0.60  +0.80  +0.800  -0.600  -01.333  -00.750  -01.667  +01.250
-0.70  +0.71  +0.714  -0.700  -01.020  -00.980  -01.429  +01.400
-0.80  +0.60  +0.600  -0.800  -00.750  -01.333  -01.250  +01.667
-0.90  +0.44  +0.436  -0.900  -00.484  -02.065  -01.111  +02.294
-1.00  +0.00  +0.000  -1.000  +00.000   UNDEF   -01.000   UNDEF
```

SUMMARY OF NBS

Abbreviations

Lower case symbols specify the following:

v	=	any variable	n	=	any number
nv	=	numeric variable	ln	=	line number
sv	=	string variable	stmt	=	any statement
e	=	any expression	x	=	any single letter or array name (without subscript)
le	=	logical expression			
ne	=	numeric expression	y	=	string array name
se	=	string expression	z	=	array name (x or y)

OPERATORS		PRECEDENCE
↑ or **	Exponentiation	1
*	Multiplication	2
/	Division	2
+	Addition or string concatenation	3
-	Subtraction or negation	3
MOD	Modulo	
<	Less than	
<=	Less than or equal to	
=	Equal to	
>	Greater than	
>=	Greater than or equal to	
<> or #	Not equal to	
<<	Very much less than	
>>	Very much greater than	
=#	Approximately equal to	
NOT	Logical complement	1
AND	Logical conjunction	2
OR	Logical disjunction	3
XOR	Logical exclusive OR	3
BUT	Logical conjunction	4
IMP	Logical implication	
EQU	Logical equivalence	
BAN	Binary conjunction (used in INTEGER mode only)	
BOR	Binary disjunction (used in INTEGER mode only)	
BEX	Binary exclusive OR (used in INTEGER mode only)	

PROGRAM STATEMENTS

These statements are listed in alphabetic order. The letters D and I at the left indicate whether statements may be used in DIRECT or INDIRECT mode respectively. Note that in NEWBASIC more than one statement may be written on a line and that square brackets may be used to group statements for execution in a special order.

D I	ACCEPT	See INPUT.
D	APPEND /file/	Merges the named file and the current program. When duplicate line numbers exist, lines in the named file replace those in the current program. After the indirect APPEND is completed, execution resumes at the next statement in sequence after the APPEND statement.
I	APPEND '/file/'	
D I	BASE n	Causes the subscripts of arrays not yet dimensioned to begin at the number specified (n) rather than at 1.
D I	CALL subroutine name	Calls a NEWBASIC subroutine.
D I	CALL FNx(argument list)	Calls a programmer-defined function-subroutine.
D I	CALL $subroutine name	Calls an XTRAN subroutine. The $ is required only if the name corresponds to a key word in NEWBASIC.
D I	CLOSE ne	Closes data file opened for input or output as file ne.
I	DATA $n_1,n_2,...$	Stores the numbers which will be used as the values of the variables named in a READ statement.
I	DEF FNx (argument list)=e	Defines a function.
I	DEF FNx(argument list) . . . FNx=e RETURN	Defines a multiple-line subroutine-function. The argument list contains an arbitrary number of variables. The function may include any number of arguments. The definition must end with (1)

(cont. next pg.)

	DEF FNx(argument list) (cont.)	a statement for returning the value of the function and (2) a RETURN statement.
D I	DEMAND	See INPUT.
D I	DIM z(ne_1,ne_2,ne_3,...)	Reserves array storage for arrays whose dimensions are given by ne_1, ne_2, ne_3,....
D I	DISPLAY	See PRINT.
I	END	Terminates a program. END should be used to separate the main program from any functions or subroutines that follow it.
D* I	FOR v=ne_1 FOR v=ne_1 TO ne_2 FOR v=ne_1 TO ne_2 STEP ne_3 FOR v=ne_1 BY ne_3 TO ne_2	Causes execution to loop through a set of operations until the conditions specified are true. Incrementation is by one unless a STEP is specified (ne_3). Note that the words STEP and BY are interchangeable. A NEXT statement terminates the FOR loop except when used as a suffix.
D* I	FOR v=ne_1,ne_2,ne_3,...	Causes execution to loop through a set of operations for the specified values.
D* I	FOR v=ne_1 STEP ne_2	Causes execution to loop indefinitely.
D* I	FOR v=ne UNTIL le FOR v=ne_1 UNTIL le STEP ne_2	Causes execution to loop through a set of operations until expression le is true.
D* I	FOR v=ne_1 WHILE le BY ne_2 FOR v=ne_1 by ne_2 WHILE le FOR v=ne_1,ne_2,...STEP ne_3 TO ne_4, WHILE le	Causes execution to loop through a set of operations until the expression specified is false.
D I	GOTO ln	Transfers execution to statement ln rather than continuing execution with the next statement in sequence.

*FOR is valid in direct mode when used as a suffix.
(See page 9-10)

D I	GOSUB ln	Transfers execution to a subprogram beginning at statement ln. The subprogram must end with a RETURN statement which transfers execution back to the statement in the main program following the GOSUB.*
I	IF le THEN ln IF le [THEN] GOTO ln IF le [THEN] GOSUB ln	Transfers execution to statement ln if the expression specified by le is true.
I	IF le THEN stmt	Executes the statement specified if the expression le is true.
I	IF le THEN ln_1 ELSE ln_2 IF le THEN ln_1 ELSE GOTO ln_2 IF le THEN ln_1 ELSE GOSUB ln_2 IF le^2THEN $stmt_1$ ELSE $stmt_2$	An IF-THEN statement may optionally be followed by an ELSE clause which will be executed if the relation specified is false. When the IF condition is true, the ELSE clause is ignored and the THEN clause is executed. Note that a THEN or an ELSE clause may also include a suffix modifier.
D I	INPUT $v_1,v_2,...$	Enters data during the running of a program. When an INPUT statement is encountered, execution halts and NEWBASIC prints a ? indicating that it is waiting for one or more values to be entered from the terminal. The values entered are assigned to the variables listed ($v_1,v_2,...$). Note that ACCEPT or DEMAND may be used instead of the word INPUT.
D I	INPUT FROM ne:$v_1,v_2,...$	Enters data from the symbolic or binary file which has been opened as unit ne. The values read from the file are assigned to the specified variables.
D I	INPUT FROM ne_1 at ne_2: $v_1,v_2,...$	Enters data from a random binary file opened as unit ne_1, starting at location ne_2 on the file.

*This sequence can be altered by using the multiple statement:

(IF le) GOSUB ln_1 GOTO ln_2

This statement transfers execution to statement ln_1. When the RETURN is encountered, transfer is then made to statement ln_2.

D I	INPUT IN FORM se:v_1,v_2,...	Used to input the values of variables or expressions in special format for use with a PRINT IN FORM statement. (See PRINT.)
D I	INPUT FROM ne IN FORM se: v_1,v_2,...	
D I	LET v=e	Assigns values to program variables. The word LET is optional. Alternate forms are available for multiple assignment [LET v_1←v_2←e] and assignment of a logical value [LET v = (ne_1=ne_2)].
I	LINK '/file/'	Deletes the current program, saving the values of the variables, and begins execution at the first statement of the specified file.
D	LOAD /file/	Deletes the current program and variables. In indirect mode it starts execution at the first statement of the program specified.
I	LOAD '/file/'	
D I	LOGOUT	Logs the user off the system. (Same as the EXECUTIVE LOGOUT command)
I	NEXT v	Terminates a loop initiated by a FOR statement. The v (ln) must correspond to the variable (line number) of the FOR statement. The line number parameter is used to identify FOR-NEXT pairs in nested loops.
I	NEXT ln	
I	NORMAL MODE IS x	Sets the implicit mode of variables to the specified (COMPLEX, INTEGER, REAL, DOUBLE INTEGER, DOUBLE REAL or STRING). Variables whose names contain a $ will remain string variables until explicitly redefined. Unless specified, normal mode is REAL.
D I	ON ne GOTO ln_1,ln_2,...	Transfers execution to statement ln, depending on the value of ne. If the integer part of ne is 1, execution transfers to ln_1, if 2, to ln_2, etc.
	ON ne GOSUB ln_1,ln_2,...	

D I	ON ENDFILE (ne) stmt	Executes the statement specified when the END OF FILE ne is read.
D I D I	ON ERROR GOTO ln ON ESCAPE GOTO ln	Transfers execution to statement ln when an error or an escape is encountered.
D I D I	OPEN "/file/",INPUT,ne OPEN /file/,OUTPUT BINARY,ne	Opens a symbolic or binary file for input or output. Delimiting quotation marks may be used with the file name but are optional in an OPEN statement.
D I D I	OPEN /file/ RANDOM INPUT ne OPEN /file/ RANDOM OUTPUT ne	Opens a random binary file for input or output as file ne.
D I D I	OPEN // FOR INPUT ne OPEN // FOR OUTPUT ne	Enables user to specify the file name during program execution.
D I D I	OPEN /file/ FOR INPUT AS FILE ne OPEN /file/ FOR INPUT, FILE ne	Opens a file for input as the named file and attaches it to unit ne. (ne must be greater than 2.)
D I	OPEN se FOR INPUT AS ne	Opens string expression se for input as ne.
D I	OUTPUT	See PRINT.
D I	PAUSE	Causes program execution to halt and control to return to the NEW-BASIC command symbol.
D I D I D I	PRINT e_1,e_2,... PRINT e_1;e_2;... PRINT e_1:e_2:...	Prints the values of expressions or variables. The comma is used for normal field output, the semicolon for packed output, and the colon for concatenated output. (WRITE, TYPE, DISPLAY or OUTPUT may be used instead of the word PRINT.)
D I	PRINT ON ne:v_1,v_2,...	Writes the values of the listed variables on a symbolic or binary file opened as unit ne.
D I	PRINT ON ne_1 at ne_2: v_1,v_2,...	Writes the values of the listed variables on a random binary file designated as unit ne_1, starting at location ne_2 on the file.

D I PRINT IN FORM se:e_1,e_2,...

Prints the values of variables or expressions in the field format specified where the string expression is:

R	Free format (decimal or scientific notation form)
D	Indicates position of digit; zero always printed.
Y	Digit; zero replaced by blank.
Z	Digit; leading zeroes replaced by blanks.
Q	Left-adjusted number; unneeded character positions are suppressed.
*	Check protect (asterisk filling)
$	Floating dollar sign.
S	Floating sign.
-	Prints sign if number is negative (floats).
+	Prints sign if number is positive (floats).
.	Positions decimal point.
V	Positions but does not print decimal point.
E	Positions exponent (forces scientific notation).
Space	Separates fields in form.
B	Prints blanks on output.
Text	Prints literally any string enclosed in primes or quotation marks.
/	Prints a carriage return.
,	Prints a comma when not the leading nonblank character.

D	I	READ $v_1, v_2, \cdots$	Assigns values obtained from the DATA list to the listed variables.
D	I	REM text	Supplies remarks for program description. Remarks are not printed except during LISTing of the program.
D	I	RESTORE	Restores the DATA list after the numbers in it have all been read, allowing it to be reread by another READ statement.
	I	RETURN	Returns execution to the next statement in sequence in the main program after a GOSUB subprogram or a multiple line programmer-defined function has been executed.
	I	STOP	Halts program execution. A direct GOTO statement can be used to restart execution.
D	I	TYPE	See PRINT.
D	I	VAR=ZERO	Sets all variables equal to zero.
D	I	WRITE	See PRINT.

MATRIX STATEMENTS

D	I	MAT $x_1 = x_2$	Replaces matrix x_1 with matrix x_2.
D	I	MAT $x = x_1 * x_2$	Multiplies matrix x_1 by matrix x_2. The same matrix name may not appear on both sides of the equation. Also, the number of rows in x_2 must equal the number of columns in x_1.
D	I	MAT $x = x_1 + x_2$	Adds matrix x_1 to matrix x_2. x_1 and x_2 must have the same number of rows and columns.

D I MAT x = x_1 - x_2	Subtracts matrix x_2 from matrix x_1. x_1 and x_2 must have the same number of rows and columns. Matrix x may appear on both sides of the equation in subtraction or addition.
D I MAT x = x_1 * (ne)	Matrix scalar multiplication.
D I MAT x = ZER D I MAT x = ZER (ne_1,ne_2,...)	Sets the working size of the matrix and sets all the elements of the matrix to zero. (This statement dimensions the matrix as x(ne_1,ne_2,...).)
D I MAT x = IDN D I MAT x = IDN (ne_1,ne_2,...)	Sets the working size of the matrix and sets up the matrix as an identity matrix. (This statement dimensions the matrix as x(ne_1,ne_2,...).)
D I MAT x = CON D I MAT x = CON (ne_1,ne_2,...)	Sets the working size of the matrix and sets all the elements of the matrix to one. (This statement dimensions the matrix as x(ne_1,ne_2,...).)
D I MAT INPUT z	Sets the working size of the matrix. Values are input during the running of the program. (Note that ACCEPT or DEMAND may be used instead of the word INPUT.)
D I MAT INPUT z(n_1:n_2,n_3:n_4)	Sets the working size of the matrix and dimensions the array as specified (n_1:n_2,n_3:n_4). Values are input during program execution.
D I MAT INPUT z FROM ne: z(n_1:n_2,n_3:n_4)	Sets the working size of the matrix and dimensions the array as specified (n_1:n_2,n_3:n_4). Values are input from a file designed as unit ne.
D I MAT PRINT z_1,z_2,... D I MAT PRINT z_1;z_2;... D I MAT PRINT z_1:z_2:...	Prints the matrices on the terminal. Commas, semicolons, or colons are used in the statement to determine the form of the output. (Note that WRITE, TYPE, OUTPUT, or DISPLAY can be used instead of the word PRINT.)

D I	MAT READ z	Sets the working size of the matrix. Values are read in from a data list.

DATA TYPES

D I	COMPLEX $v_1,v_2,...$	Declares the specified variables to be in COMPLEX mode.
D I	DOUBLE INTEGER $v_1,v_2,...$	Declares the specified variables to be in DOUBLE INTEGER mode.
D I	DOUBLE REAL $v_1,v_2,...$	Declares the specified variables to be in DOUBLE REAL (double precision) mode.
D I	INTEGER $v_1,v_2,...$	Declares the specified variables to be in INTEGER mode.
D I	REAL $v_1,v_2,...$	Declares the specified variables to be in REAL mode.
D I	STRING $v_1,v_2,...$	Declares the specified variables to be in STRING mode. Text is equivalent to STRING mode.

SUFFIX MODIFIERS

(Note: A logical expression is considered false if it is equal to zero and true if it is not equal to zero:)

D I	stmt FOR nv=$ne_1,ne_2,...$ D I stmt FOR nv=ne_1 TO ne_2 STEP ne_3 D I stmt FOR nv=ne UNTIL le D I stmt FOR nv=ne WHILE le BY ne_2	Causes execution to loop through a set of operations for a specified range of values or until a logical condition is met. (Additional syntactic forms of FOR are shown in the PROGRAM STATEMENTS section.)
D I	stmt IF le	Causes the statement to which it is appended to be executed if the logical expression le is true.
D I	stmt UNLESS le	Causes the statement to which it is appended to be executed only if the logical expression i.e. is false.

D I	stmt UNTIL le	Causes the statement to which it is appended to be executed repeatedly as long as the logical expression le is false.
D I	stmt WHILE le	Causes the statement to which it is appended to be executed repeatedly as long as the expression le is true.

OPERATING COMMANDS

D	AGAIN	Repeats the last statement executed and resumes program execution.
D	APPEND /file/	Merges the named file and the current program. When duplicate line numbers exist, lines in the named file replace those in the current program.
D	BREAK ln,ln,...	Sets breakpoints at the specified lines.
D	CONTROL-G	Used to escape out of program execution at an INPUT request or to return to the EXECUTIVE from the NEWBASIC command symbol.
D	DELETE ln, ln-ln	Deletes the specified line or range.
D	DUMP /file/	Creates a core image of the current program and stores it in compiled binary from under the name specified. DUMP files may only be executed or deleted. They are executed by issuing the command EXECUTE /file/ to the EXECUTIVE command symbol or by typing just the file name (/file/) to the EXECUTIVE dash.
D	EDIT ln	Types out the specified line and makes it the previous line for editing purposes.

D	EXTRACT ln, ln-ln	Deletes all but the specified lines.
D	EXIT	Used to exit from a subsystem.
D	LIST LIST ln, ln-ln	Lists the current program. Individual line numbers or ranges of lines can be specified as parameters to the command.
D	LISTNH LISTNH ln, ln-ln	Same as list, except that the heading containing the date, time, and program name are not typed at the top of the listing. (LISTNH may not be abbreviated as other commands.)
D	LENGTH	Types the length of the current program in lines.
D	LOAD /file/	Retrieves a previously SAVEd file and places it in working storage as the current program. (Same as OLD)
D	MODIFY ln	Locates the specified line and makes it the previous line for editing purposes. The line is not typed out.
D	NBS	Prints OK? Upon confirmation, deletes all of program and variables.
D	NEW /file/	Creates a new file with the name specified.
D	OLD /file/	Retrieves a previously SAVEd file and places it in working storage as the current program.
D	POINTS	Lists all breakpoints currently being used.
D	PROCEED	Restarts program execution after a breakpoint.
D	QED	Transfers the current file into the QED subsystem for further editing. Typing a G^{c} (or the QED command SCO) transfers the edited program back into NEWBASIC.

D	RECOMPILE	Recompiles a program after correcting an error which caused NEWBASIC to request a nonexistent subprogram.
D D	RENUMBER RENUMBER ln, ln-ln, n	Resequences line numbers in the current file starting with line 100 and incrementing by 10. The starting line number, the range of lines to be resequenced, and the increment to be used can be supplied as parameters to the command.
D D	RUN RUN /file/	Begins execution of the current program or the file named in the command. All variables are set to zero when the command is issued.
D	SAVE /file/	Saves a permanent copy of the current program on the file specified.
D	SCRATCH	Deletes the current program but saves the file name and the contents of variable storage.
D	STEP	Executes one statement after a breakpoint and then halts again (may be abbreviated as S).
D	TABS n,n	Sets tabs at the specified points for special tabbing requirements. Standard tab positions are set at 8, 16, 32, 40, 48, and 56.
D	TAPE	Appends a file or series of line-numbered NEWBASIC statements from paper tape.
D	UNBREAK	Removes all breakpoints.
D	UNBREAK ln,ln,...	Removes breakpoints from the specified lines.
D	WHERE	Lists the line number of the next statement to be executed.
D	WIDTH n	Sets the assumed width of the terminal carriage to a number other than 72 columns.

EDITING CONTROL CHARACTERS

A^C	Deletes the previous character (echoes ↑)
C^C	Copies one character
D^C	Copies through end of line
E^C	Begins or ends insertion (echoes < or >)
F^C	Copies through end of lines; does not print at terminal.
H^C	Copies up to end of line
I^C	Tabs to the next tab stop position
K^C	Restarts edit, saving changes
O^C x	Copies up to specified character x.
Q^C	Restarts edit; does not save changes.
R^C	Retypes line to current point.
S^C	Skips single character from input (echoes %)
U^C	Copies line to next tab stop.
V^C x	Accepts next character literally.
W^C	Deletes preceding word (echoes \)
X^C x	Skips characters through specified character (echoes %)
Y^C	Skips line to next tab stop (echoes %)
Z^C x	Copies through specified character x.

NUMERIC FUNCTIONS

ABS(X)	Absolute value of X
INT(X)	Integer part of X
MOD(X,Y)	X modulo Y
SGN(X)	Sign of argument X
DIF(X,Y)	Positive difference ABS(X-Y)
EXP(X)	Exponential of X
LOG(X)	Natural log of X
LGT,LOG10(X)	Log, base 10, of X
SQR,SQRT(X)	Square root of X
SIN(X)	Sine of X
COS(X)	Cosine of X
TAN(X)	Tangent of X
ARCSIN(X)	Arcsine of X
ARCCOS(X)	Arccosine of X
ATAN(X),ATAN(X,Y)	Arctangent of X or of X/Y
SINH(X)	Hyperbolic sine of X
COSH(X)	Hyperbolic cosine of X
TANH(X)	Hyperbolic tangent of X
FIX(X)	Integer mode of X (truncates)
FLOAT(I)	Floating point mode form of I
SNGL(D)	Single precision mode form of D
NUM(X)	Random integer from 1 to x
LSH(I,J)	Binary left shift I for J positions
RSH(I,J)	Binary left shift I for J positions

IMAG(C)	Imaginary part of complex number C
REAL(C)	Real part of complex number C
CMPLX(X,Y)	Complex number X,Y
CONJG(C)	Conjugate of (X,Y) = (X,-Y)
MAX(X,...,Z)	Maximum of arguments
MIN(X,...,Z)	Minimum of arguments
WAIT (X)	Halts execution for X seconds
POS(I)	Reads position of file I
PASS PASS (ln)	Number of times statement is executed.
RESPASS RESPASS (ln)	Resets pass counter
DATE	Returns a 12-character string containing the date and time.
TEL	Returns zero if the terminal input buffer is empty; otherwise it returns one.
TIME	Reads the system clock in 1/60 of a second units.

STRING FUNCTIONS

INDEX(se_1,se_2)	Position of se_2 within se_1; e.g., INDEX("ABC","C") = 3.
LEFT(se,ne)	Substring of se, ne characters long starting from left.
LENGTH(se)	Length of se.
RIGHT(se,ne)	Substring of se; ne characters long, starting at right.
SPACE(ne)	String ne spaces long.
STR(ne)	String of the characters comprising ne; e.g., STR(4) = "4".

SUBSTR(se,ne_1,ne_2)	Substring of se; ne_2 characters long, starting at ne_1th character.
VAL(se)	Numeric value of se, where se must be a numeric string; e.g., VAL("+8") = 8.
ASC(se)	ASCII binary equivalent of the first character of se.
CHAR(ne)	One-character string which is the ASCII character whose numeric value is ne.
CTI(se)	Internal binary equivalent of first character of se.
ITC(ne	One-character string which is the character whose internal ASCII value is ne.

CATALYST FUNCTIONS

ICO(R$,A$,K)	R$ contains the string A$ (K = 0). R$ contains the word A$ (K = 1).
IS(R$,A$,K)	R$ is the string A$ (K = 0). R$ is the word A$ (K = 1).
IBEF(R$,A$,K,B$,K)	R$ contains A$ before B$.
IEQIV(R$,A$,K)	R$ contains one of the strings in A$ delimited by commas.
CALL REIN	Reinforce.
CALL RRIN	Really reinforce.
PASS	
RESPASS	Reset PASS.
CALL REP	Prints "??--PLEASE RESPOND AGAIN"
@NBS	